

Received April 20, 2020, accepted May 21, 2020, date of publication May 26, 2020, date of current version August 27, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2997669

# NATRA: Network ACK-Based Traffic Reduction Algorithm

**SANTIAGO GARCIA-JIMENEZ<sup>1</sup>**, (Member, IEEE), **EDUARDO MAGAÑA<sup>1</sup>**, (Member, IEEE),  
**AND JAVIER ARACIL<sup>2</sup>**, (Senior Member, IEEE)

<sup>1</sup>Department of Electrical, Electronic and Communications Engineering, Universidad Pública de Navarra, 31006 Pamplona, Spain

<sup>2</sup>Department of Electronic and Communication Technology, Universidad Autónoma de Madrid, 28049 Madrid, Spain

Corresponding author: Santiago García-Jimenez (santiago.garcia@unavarra.es)

This work was supported by the Spanish Ministry of Economy and Competitiveness and the European Regional Development Fund through the projects TRAFICA under Grant MINECO/FEDER TEC2015-69417-C2-1-R and Procesado Inteligente de Tráfico under Grant MINECO/FEDER TEC2015-69417-C2-2-R.

**ABSTRACT** Traffic monitoring involves packet capturing and processing at a very high rate of packets per second. Typically, flow records are generated from the packet traffic, such as TCP flow records that feature the number of bytes and packets in each direction, flow duration, number of different ports, and other metrics. Delivering such flow records, about network traffic flowing at tens of Gbps is rather challenging in terms of processing power. To address this problem, traffic thinning can be applied to reduce the input load, by swiftly discarding useless packets at the sniffer NIC or driver level, which effectively reduces the load on software layers that handle traffic processing. This work proposes an algorithm that drops empty ACK packets from TCP traffic, thus achieving a significant reduction in the packets per second that must be handled by each traffic module. The tests discussed below show that the algorithm achieves a 25% decrease in the packets per second rate with minimal information loss.

**INDEX TERMS** Network traffic thinning, traffic processing, sniffer architecture.

## I. INTRODUCTION

To meet increasing bandwidth demands, Internet providers are deploying high-speed lines across the network. Core providers such as Internet2 and ESnet have upgraded their link rates up to 100 Gbps, and the same applies to regional networks [38].

Traffic monitoring probes are typically deployed in such networks to assess quality of service and reinforce network security. Such traffic probes produce enriched flow records on the fly, which can then be used to deliver Key Performance Indicators (KPIs) about networks, systems, and applications. Because TCP traffic accounts for most Internet traffic [10], [15], [16], [21], [25], a great number of quality-of-service KPIs are related to TCP connection parameters. For example, a high number of duplicated ACKs maybe indicative of packet loss. Thus, enriched TCP connection records that include information about retransmissions and other features are needed.

In this challenging scenario, capturing and analysing every packet in the network for traffic analysis is no longer

feasible [34]. Capturing every single packet flowing through the network will result in a practically intractable volume of data. Furthermore, the cost of sniffers will be very high, in terms of both processing power and storage requirements [42]. Such a large data rate and information volume calls for a radical departure from the traditional approach of capturing every piece of data and analysing them later. Selective packet capture is sure to be foundational in the analysis of traffic rates of 40 Gbps and beyond [6], [17], [22].

Selective packet capture implies that some information will be lost in flow records; accuracy and performance are in a trade-off relationship. We note that the performance metric for nearly all packet sniffers is the data rate that they can handle in packets per second. At the same bits-per-second (bps) rate, smaller packet sizes increase the load in packets per second. Each incoming packet is an event that the packet sniffer must handle. As packets per second increase, the sniffer must work harder to cope with the traffic load.

In light of the above concerns, we advocate for traffic thinning techniques that decrease the sniffer load in terms of packets per second, without serious information loss. Such traffic thinning techniques can be implemented in the sniffer's Network Interface Card (NIC), immediately after the packet

The associate editor coordinating the review of this manuscript and approving it for publication was Anton Kos<sup>1</sup>.

is captured and before it is passed to upper layers for traffic analysis. Alternatively, thinning can be implemented in the network processor hardware which allows for a degree of programmability at the hardware level [4], [7], [8], [12]. As a workaround, the latter techniques can also be implemented in the device drivers. Indeed, modifications to device driver software have proven to be useful in achieving 10-Gbps capture rates with commodity hardware [9], [17], [27], [40].

This paper proposes an algorithm that selectively removes ACK packets to reduce the sniffer data workload. By doing so, we can reduce the packets per second by a remarkable 25%. Some information might be lost, as ACKs are useful for quantifying retransmissions especially in case of duplicate ACKs. Nevertheless, retransmissions can also be estimated from the retransmitted packets themselves, which will appear in the traffic stream in the reverse direction. With this workaround, the overall information loss is minimal and is worth the resulting gains in performance.

We also note that thinning the traffic stream increases the sniffer load potentially leading to packet loss. Our proposed technique was developed to minimize machine instructions and to avoid moving data to memory from the CPU cache.

We have named our proposed traffic thinning algorithm NATRA, for “Network ACK-based Traffic Reduction Algorithm”. We propose several options for NATRA implementation, such as placing it between the NIC and the Storage Performance Development Kit (SPDK) libraries [35], [36]. This option can be used to store packets in high-speed hard disks such as non-volatile memory express (NVMe) disks. The results of this research can help designers of high-speed packet sniffers to develop cost-effective implementations. We also discuss a prototype software implementation that is ready for practical use.

The rest of this paper is organized as follows. In section II we give an overview of the design of high-speed traffic probes, and discuss potential bottlenecks in the traffic flow. Then, in section III, we introduce the NATRA algorithm. Section IV discusses performance evaluation and implementation. Section IV-A presents an analytical model that allows us to extrapolate results to arbitrarily large data rates. Section V presents the NATRA performance evaluation results and discussion and Section VI draws the conclusions.

## II. PACKET SNIFFER SYSTEMS

This section discusses the architecture of typical packet sniffer systems, to highlight the bottlenecks that limit the system’s performance. We conclude that the packet arrival rate, in packets per second, has a great effect on overall system performance. Thus, selective packet capture can help to alleviate the packet packet sniffer’s workload and allows us to increase the range of network bandwidths for which the sniffer is effective.

Packet capture takes place in the NIC, which can either be a network processor [19] or a general-purpose device [28]. The NIC usually receives traffic from a SPAN port or splitter. The driver manages the load, balancing the traffic analysis among

several processing cores to generate flow records in the form of time series or aggregate statistics. Finally, the outcome of traffic analysis (and/or the raw packet trace) is passed to permanent storage, typically using a disk array (RAID), with SATA, SSD or even NVMe disks for traffic storage at 40 Gbps or more. We note that load balancing in cores can take advantage of Receive Side Scaling (RSS) techniques [37], which split the packet stream into several queues at the NIC. Then, the traffic from each queue is handled by a different processing core, achieving parallel traffic processing.

The state of the art in high-speed packet capturing shows significant advances in the area, specially in SmartNICs. Such NICs provide spare processing resources so as to alleviate the processing load at the host CPU [23], [32]. Trumpet [29] leverages DPDK technology to provide fast triggers and statistics on 10 Gbps networks. Flowatcher [41] is also based on DPDK and provides statistics on packet and flow level reaching speeds up to 10 Gbps. Flowscope [14] enables storing high-speed traffic up to 100 Gbps for subsequent processing. We note that all these approaches are complementary to the research presented in this paper, as they focus on high-speed traffic capturing, but not thinning. Indeed, NATRA could also be implemented at the *SmartNIC* level.

In any case, we distinguish several steps in the process for packet capture and analysis as seen in Fig. 1:

- First, the NIC must copy the packet from the network into its internal memory and sometimes the NIC will apply pre-processing like RSS or packet filtering. These steps are usually performed in hardware at the line rate, and no packet loss should happen at that point.
- Second, the driver handles NIC interrupts and relays the packets from the NIC internal memory to kernel memory through DMA cycles. Clearly, as the interrupt rate increases this copying task becomes more difficult. Hence, interrupt-coalescence techniques are used to reduce the processing burden at the interface between the NIC and the driver [24]. If the interrupt rate exceeds the driver capacity, packets will be lost.
- Third, the processing cores copies the packets from kernel memory into user space and perform the analysis. The first step is extracting protocol fields like the IP addresses or VLAN tags. This step is time-consuming, especially with variable field headers, and requires a high CPU speed. To achieve greater throughput, CPU parallelism is normally exploited using receive-side scaling (RSS). Thus, many cores are required at the sniffer, which increases hardware costs. Because the CPU is limited in terms both of speed and number of cores packet loss may also happen at this stage.
- Fourth, the outcome of this analysis (flow records, packet captures, etc) must be dumped to permanent storage. Records storage is also limited by read/write speed, even though this speed has increased dramatically with the recent introduction of SSD and NVMe technologies. Ultra-fast storage is very expensive, however, and the sniffer storage needs are large. Thus, speed

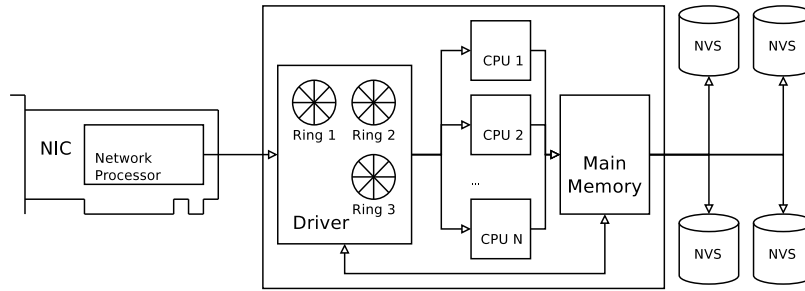


FIGURE 1. Flow chart for packet capture.

and storage volume are in a trade-off relationship and both are required for high-speed packet capture. Indeed, storage is the most serious bottleneck in the process and may also lead to packet loss.

At any of the above stages, traffic thinning can reduce the load at the subsequent stage and thereby reduce the cost of sniffers. The next section presents the NATRA algorithm, which is specifically designed to reduce the sniffer workload.

#### A. TCP CONNECTION RECORDS

Before further discussion of the algorithm design and implementation, we briefly discuss the dynamics of flow record generation in a packet sniffer. TCP packets are used by the packet sniffer to obtain flow records that required to present KPIs in dashboards. These TCP flow records include several fields which can be grouped in the following categories:

- *Volume*: TCP flow size and duration, which are helpful for calculating the traffic per port, IP addresses, and other indicators.
- *Congestion indicators*: Zero window announcements from the client and server can indicate a possibly saturated host.
- *Flags*: SYN or RST flags help to identify connection attempts with not response, abrupt resets, and other interruptions.
- *RTT*: can be measured as the time difference between sending a SYN packet and receiving the first ACK packet as shown in Figure 2.
- *Loss indication*: can be estimated from the number of observed retransmissions and the number of duplicated ACKs.

To obtain the above indicators, a hash table that contains one entry per TCP flow (or two entries for the two unidirectional flows that comprise a connection) is stored in memory. These entries are updated with all incoming TCP packets that belong to the flow. Most importantly, the processing cost is not equal for all the above performance indicators, which practically means that not all the indicators can be obtained online.

Allocating memory to keep record of ACKs, at 40 Gbps or more is a challenge in itself. Complicating the issue, the interarrival time between a packet and its matching ACK can reach 500 ms [5], with the delayed ACK (200 ms in a

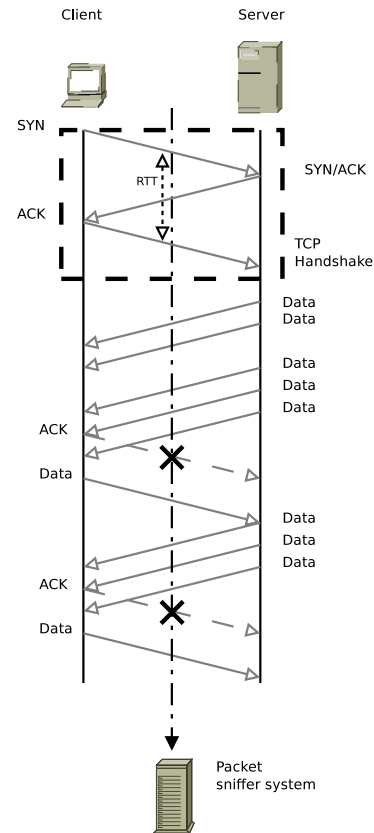


FIGURE 2. TCP handshake and RTT estimation.

Windows operating system [39]). Consequently, the number of records in memory grows accordingly and the lookup time needed for finding a matching packet severely degrades the sniffer performance.

#### III. NATRA TRAFFIC THINNING ALGORITHM

The proposed traffic thinning algorithm achieves a significant reduction of the TCP packet arrival rate by removing ACK packets without payload. We focus on thinning TCP traffic, because most Internet traffic uses TCP, as can be seen in figure 3. This figure shows the percentage of TCP traffic over total traffic recorded in CAIDA data from 2002 to 2018 [10]. The alternative transport protocol alternative QUIC [20], works over UDP layer and improves the communication

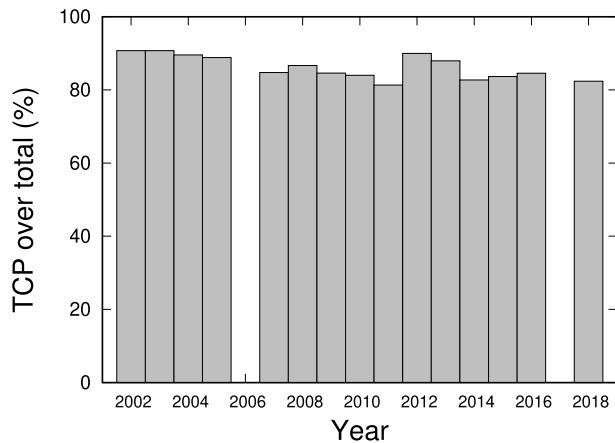


FIGURE 3. TCP over total traffic on CAIDA traces.

strategy for connection-oriented web applications, but TCP remains as the main protocol used in network trunk lines.

By thinning TCP traffic, we expect to obtain significant savings in the overall packet arrival rate. In what follows, we use the term ACK to denote packets that carry no data payload and serve only to acknowledge the arrival of packets from the other end of the connection.

#### A. INFORMATION LOST WHEN REMOVING ACK PACKETS

This section discusses the information that is lost when removing ACKs and assesses the importance of such information loss compared to the resulting performance gains.

Next, we discuss how KPIs will be affected by the removal of ACKs. We note that the main two affected parameters are the estimation of retransmissions (including duplicated ACKs) and RTTs. As for the former, ACKs are useful to determining whether a packet has been lost, which is typically indicated if several ACKs arrive with acknowledgment numbers smaller than the last sequence number seen in the same TCP flow. As an alternative, we can use the retransmission itself, which will eventually be triggered by the transmitter. Packet loss can also be detected from holes in the stream of sequence numbers. In this case, we should consider that such time spaces can also arise if packets arrive out of order.

We further note that in practical analysis only TCP connections with a significant number of retransmissions are relevant. For retransmitted packets, the sequence number interval (sequence number, sequence number + length) will overlap with that of the original transmitted packets.

Therefore, the corresponding TCP connections can be detected easily by counting retransmitted packets instead of duplicated ACKs [26].

RTT is a significant KPI that should not be lost. RTT can help in detecting network paths subject to considerable latency. Clearly the estimation of RTT will be affected seriously by the removal of ACKs. At this point, we propose an alternative that allows us to estimate the RTT per connection by saving only the first ACK-only packet of the connection. The implementation of this alternative design involves

tracking the connection state only during the connection establishment phase, which can be expensive in terms of processing. Even so, we have obtained a remarkable speed of 31.45 Gbps per core with an i5-4570 3.20-GHz CPU, in our software implementation for tracking the connection establishment phase.

The first technique presents a trade-off between the accuracy of RTT estimation and performance gains in terms of packets per second due to ACK removal. In practice, the online estimation of all the TCP RTTs in the connection is very processing intensive and the problem is usually solved with offline analysis at speeds of many Gbps. Moreover, RTT estimation can also be performed using the ACK packets that do carry payload, which will not be removed. The proposed technique for RTT estimation in the connection handshake along with the samples produced by ACK packets with payload is sufficient for the retaining the KPI of RTT.

#### B. NATRA THINNING PROCESS

Our proposed thinning process drops all packets that carry the ACK flag and no data *except those that participate in the handshake that establishes the TCP connection* (see Fig. 2). To this end, we only keep track of the state of the connection establishment phase during the handshake for all TCP flows. This process includes the first SYN packet from the client, the SYN+ACK packet from the server and the first ACK from the client. Then, the handshake RTT will be calculated using this information.

We will use different real traffic traces to measure the reduction provided by the proposed method. For the sake of generality, six different traces were used, as shown in Table 1. The first one was taken from a lab in our university and it has a low flow rate and a heterogeneous mix of traffic. This trace mostly includes traffic generated by the academic web applications used by students and teachers, in addition to traffic involved in standard Internet navigation. The second trace was recorded from the traffic at a large company; it includes traffic from office and business applications. In addition, we used three more traces from the CAIDA repository [11]. This last recorded by the CAIDA group from a central segment of the Internet and it is publicly available; this will ensure the reproducibility of the results. Hence, the three traces give a balanced mix of academic, industrial, and public Internet traffic. The three CAIDA traces cover an hour worth of traffic through the Equinix-Chicago link.

Finally, The last trace is from a university Internet link. To be able to replay real network traffic, we need traces with a defined MAC level, non caplen, non anonymous addresses, and with playable bandwidth.

Table 1 shows the reduction achieved when using NATRA. The total reduction in packet ranges between 18.7% and 34.54%, depending on the trace. the difference in reduction ratios between traces depends on the number of ACK packets without payload. We note that traces that include applications with a high level of interactivity or, generally speaking, TCP

TABLE 1. Trace datasets.

Trace	Gigabytes	Gigapackets	Duration	TCP Flows number	Packet rate reduction
University laboratory	257	0.08	15.34 Hours	37,916	20.68%
Private company	1,651	2.3	7 Days	11,290,743	34.54%
CAIDA 2014 equinix-chicago	2,444	3.2	1 Hour	29,125,747	25.46%
CAIDA 2015 equinix-chicago	2,781	3.5	1 Hour	42,947,015	18.70%
CAIDA 2016 equinix-chicago	2,961	4.1	1 Hour	50,601,422	25.79%
University Internet link	136	0.15	2500 seconds	1,256,096	29.39%

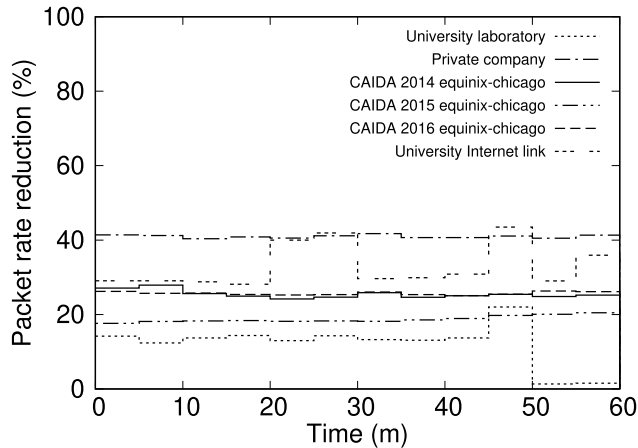


FIGURE 4. Packet-rate reduction for 1 hour of each packet trace (aggregation interval 5 min).

flows with data transfer in both directions have fewer ACK packets.

Figure 4 shows the reduction in traffic that we could achieve during a full hour in each trace. We selected 1 hour from each dataset to all traces in similar form, because the CAIDA traces have one hour of data. Data was aggregated in intervals of 5 minutes. Different datasets provide different reduction rates. We obtained rates close to 40% when testing the private-company traces, which is the best performance overall. The packet reduction is stable over time for any of the networks.

Additionally, we have also assessed the impact in a large datacenter from a banking company. The total packet reduction is very similar to the reduction observed on the previous traces. We observe a potential reduction of 36.5% over the TCP traffic and 27.36% over total traffic. The percentage of TCP packets over the total is 71%, also similar to the one observed on CAIDA traces.

Overall, the experimental results indicate that NATRA shows promising performance in a variety of traffic scenarios, reducing traffic by nearly 25%. Traffic reduction can reach nearly 40% depending on the traffic characteristics and the specific hour in which the data was recorded.

As it turns out, QUIC is gaining importance in datacenter traffic, as it will be adopted for HTTP3. Even though traffic thinning for QUIC is out of the scope of this paper we perform an exploratory analysis of packet sizes in Figure 5, which

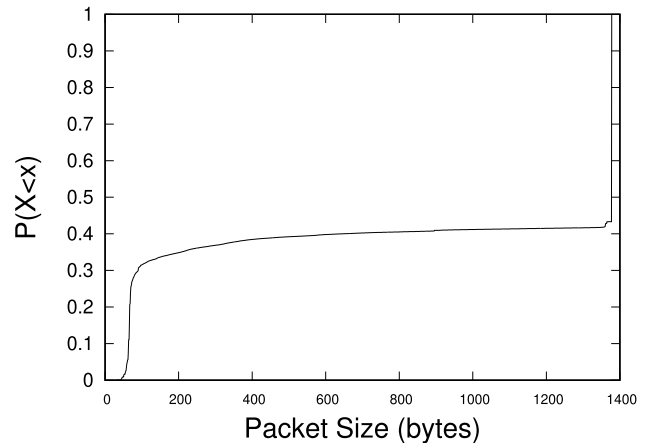


FIGURE 5. Packet size CDF on QUIC protocol.

shows QUIC packet size CDF from traffic of our University access link. Interestingly, 30% of packets are shorter than 90 bytes, and 56% of packets are larger than 1378 bytes. Despite QUIC data is encrypted, packets sizes could be used as a way to differentiate between data and acknowledgement packets. This finding opens new research avenues for QUIC traffic thinning, that are left for the sequel.

#### IV. NATRA DESIGN AND IMPLEMENTATION

In this section, we provide insights into the design and implementation of NATRA. Briefly, NATRA drops all ACK packets that carry no data payload excepting the first packet sent from server to client (which also has the SYN flag) and the second ACK sent from client to server. To identify the first ACK, a connection record must be kept in memory that includes the connection five-tuple (source and destination IP addresses, ports, and sequence number) and the state of the connection establishment handshake. Whenever an empty data payload packet arrives, the list of connection records is searched for the corresponding match and the algorithm decides whether to drop or retain the packet.

More specifically, a static table is used to store all the unfinished TCP handshakes, which are indexed by a hash function. The space in which we will store the information is called a slot and by using different indices, we will be able to access the information stored in different slots. The hash function is the XOR function of the five-tuple modulo the theoretical maximum number of concurrent flows. The latter is estimated



from the connection interarrival time and the maximum RTT value, as will be described in section IV-A.

All the necessary input data can be obtained from the IP and TCP headers (source and destination IP and port and TCP sequence number) using a simple structure. The timestamp can be obtained from a call to the system time libraries. Ideally, the timestamp should be kept in cache memory, as it is accessed every time a new packet arrives.

Pseudocode 1 presents the algorithm that executes traffic thinning. The Check\_packet function takes a packet and a timestamp as input and returns a boolean that represents the decision about dropping the current packet. When a SYN packet arrives, the corresponding connection record is updated with the new connection establishment state (lines 11-14). The hash used to update the information will contain the TCP sequence number incremented by one unit (line 3). Furthermore, when an ACK packet with empty data payload arrives, the algorithm checks whether it corresponds to the connection establishment phase to update the RTT calculation (line 25). If the ACK-only packet belongs to the handshake, the hash will be calculated with its original sequence number so the hash will be the same as the one processed previously by using the SYN-only packet that opens the same handshake (line 19). If any of the previous connection establishment phase packets have been lost, the connection record is garbage, and must be removed after an expiration timeout. This expiration timeout should be as close as possible to the maximum RTT observed in the trace (line 5).

#### A. SYSTEM MODEL AND MEMORY REQUIREMENTS FOR HIGH-SPEED DEPLOYMENT

The connection records list is searched for all empty payload packets being received. As shown in the previous section, this connection record must be kept in memory for at least the duration of the connection RTT. At high speed, the search operation acts as a bottleneck and rapid-access memories are required, namely processor caches or on-board NIC memories. To increase performance, we choose to store the above hash tables in static memory, thus saving the time consumed by allocation and deallocation of dynamic memory. In this section, we focus on the memory requirements in terms of size and speed. Before we investigate these requirements, let us briefly discuss current cache technologies, to explain the hardware limitations. We will then calculate the amount of memory required and discuss whether it fits in typical cache memory sizes.

##### 1) CACHE TECHNOLOGIES

Table 2 lists the I/O throughput for most typical processors (in Bytes per second, according to the *Sisoftware* benchmarks [33]). The table shows the limitations in size and speed of the cache memories at different cache levels. We note that cache memory is much faster than the system RAM and the associated core has exclusive access to it, instead of sharing the RAM between the different cores. Therefore, our NATRA

#### Pseudocode 1 NATRA Packet Thinning Algorithm

```

1: function check_packet(packet, timestamp)
2:   if (packet.SYN && not packet.ACK) then
3:     hash = HASH(packet{src_ip, dst_ip,
4:       src_port, dst_port, sequence + 1})
5:     if (data[hash].used &&
6:       data[hash].timestamp - timestamp
7:       > RTT) then
8:       data[hash].used = FALSE
9:     end if
10:    if (data[hash].used == FALSE) then
11:      data[hash].packet =
12:        packet{..., sequence + 1};
13:      data[hash].timestamp = timestamp;
14:      data[hash].used = TRUE;
15:      return FALSE;
16:    end if
17:    else if (packet.ACK && not packet.SYN &&
18:      not packet.FIN && packet.data == 0) then
19:      hash = HASH(packet{src_ip, dst_ip,
20:        src_port, dst_port, sequence})
21:      if (data[hash].used == TRUE) then
22:        if (timestamp - data[hash].timestamp
23:          > RTT) then
24:          data[hash].used = FALSE;
25:        else if (data[hash].packet ==
26:          packet{..., sequence}) then
27:            data[hash].used = FALSE;
28:            return FALSE;
29:          end if
30:        end if
31:        return TRUE;
32:      end if
33:      return FALSE;
34:    end function

```

implementation strives to fit the connection records list into the level-3 cache if possible. To this end, we use a static hash table to store all opened handshakes. As will be shown later, our experimental and analytical results show that 4.99 MB are needed for the CAIDA trace replayed at 40 Gbps and 37.49 MB for 100 Gbps. Therefore, the rate of 40 Gbps *per core* can be sustained with current cache technologies. Using the data from table 2, the predicted speeds *per core* are 36.66, 42.66, 21.33, and 40.00 Gbps for the Intel i9 7900X, AMD Ryzen, Intel i7 6700K and Intel i7 5820K respectively. Most importantly, we stress that the previous calculations are *per individual cores* and that RSS techniques can be employed to demultiplex the traffic stream into several cores.

##### 2) MEMORY AND APPROXIMATE CAPACITY PLANNING

Rapid access memories are typically small, which limits the size of the list of connection records. As the network bandwidth increases, so does the number of concurrent

**TABLE 2.** Different cache level memory speeds and sizes.

CPU model	Characteristic	Cache level 1	Cache level 2	Cache level 3
Intel i9 7900X (Skylake-X)	Speed	2,200 GBps	1,010 GBps	-
	Size	10x32kB	10x1MB	13.75MB
AMD Ryzen 1700X	Speed	727 GBps	557 GBps	392 GBps
	Size	8x 32kB	8x 512kB	2x 8MB
Intel i7 6700K (Skylake)	Speed	878 GBps	402 GBps	247 GBps
	Size	4x 32kB	4x 256kB	8MB
Intel i7 5820K (Haswell-E)	Speed	1,150 GBps	500 GBps	205 GBps
	Size	6x 32kB	6x 256kB	15MB

connections and the size of the corresponding list of connection records. Thus, memory size acts as a limiting factor that restricts the applicability of NATRA in high-speed environments. In this section, we analyse the size of the list of connection records, from both empirical and analytical standpoints. The empirical analysis is intended to extrapolate the results to higher speeds.

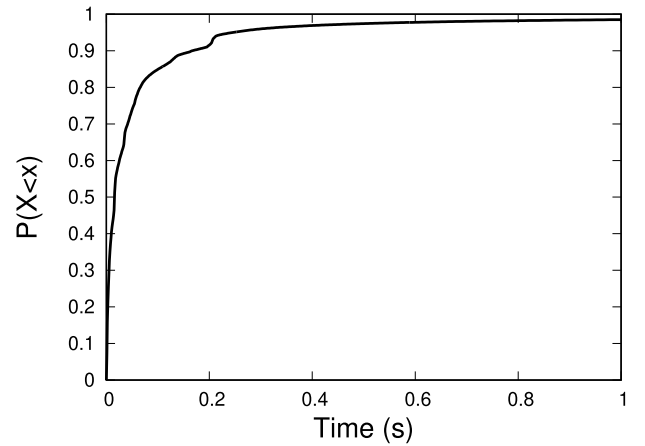
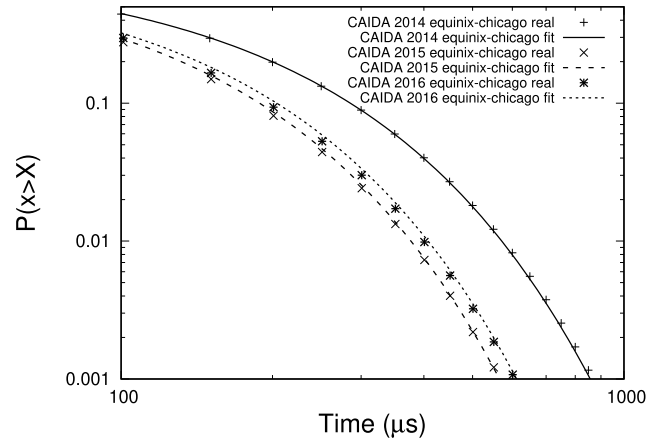
Next, we provide a model that estimates the size of the list of connection records for the sake of planning memory capacity. If the trace includes many multiplexed TCP connections, newly arrived TCP connections (whose arrivals times are assumed to follow a non-homogeneous Poisson distribution), will open a new connection record in the list. Then, the connection record will be kept open for either the RTT of the TCP handshake or the garbage-collection timeout. Consequently, the list can be modelled as an  $M/D/\infty$  queuing system, with the arrival rate of  $\lambda$  connections per second and holding time of  $1/\mu$  seconds.

This holding time for the connection record is equal to the RTT of the TCP handshake and we set an upper bound of 0.2 s, in accordance with previously reported data [31]. The selected upper bound is supported by the RTT obtained using Tshark ([30]) to derive the RTT of the TCP handshake in the trace of the university Internet link (Fig. 6).

To better understand the arrival of flows, we verify the Poisson hypothesis using the CAIDA traces, which include many multiplexed TCP connections. This analysis considers that the arrival rate changes with the time of day. Figure 7 shows the flow interarrival time distribution for the CAIDA traces. The data rate is quite stable for the one-hour duration of each CAIDA trace. As shown, the exponential distribution fits the experimental data remarkably well.

According to the model, the maximum number of concurrently open TCP handshakes in connection records follows a Poisson distribution with the parameter  $\frac{\lambda}{\mu}$ .

To calculate the maximum number of open TCP handshakes, we use the 0.99 percentile of the Poisson distribution. To this end, we employ the Normal approximation of a Poisson distribution [13]. As a result, we obtain the following formula for the number of open handshakes, denoted by  $K$ , in terms of the number of TCP connection entries in the

**FIGURE 6.** tshark RTT cumulative density function for the university trace.**FIGURE 7.** CAIDA equinix-chicago traces flow interarrival time and fit to exponential distribution.

NATRA system:

$$K = \frac{\lambda}{\mu} + z_{99} \sqrt{\frac{\lambda}{\mu}} \quad (1)$$

where  $\lambda$  is the flow arrival rate in flows per second and  $z_{99}$  is the 0.99th percentile of the standard Gaussian distribution.

We note that the hash function produces a random share of the incoming connection records such that the load per slot is  $\frac{K}{M}$  where  $M$  is the number of slots in our hash table and  $K$  follows equation 1. Thus, for any given collision probability  $P$ , the number of required slots is given simply by:

$$M = \frac{\left(\frac{\lambda}{\mu} + \sqrt{\frac{\lambda}{\mu} z_{99}}\right) (1 - P)}{P} \quad (2)$$

This theoretical approximation closely matches the experimental results as will be demonstrated in the next section. Additionally, we note that in case of a collision in the hash values (i.e. two different connections with the same hash), the corresponding connection record will not be updated, even if table of connection records has available slots. In that case, we choose not to drop the ACK packets from the colliding flow, which slightly increases the rate of packets per second but achieves a significant gain in accuracy.

## V. RESULTS AND DISCUSSION

First, we assess the validity of the above model for open handshakes, following equation 1. Then, we perform a throughput performance assessment with NATRA use cases, which also includes analyzing NATRA performance as a traffic thinning middleware for well-known traffic analysis tools.

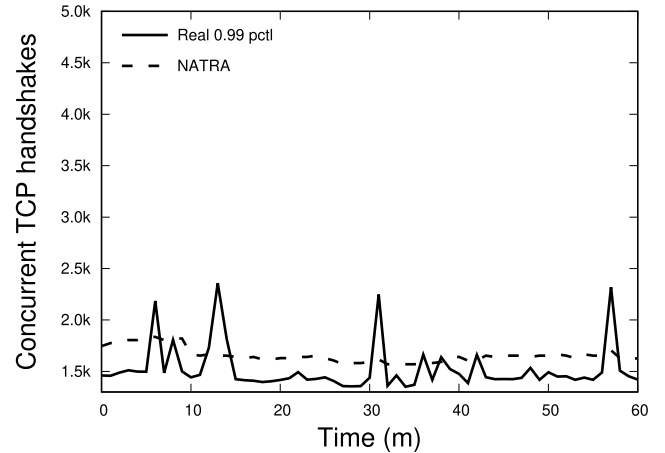
The results for the CAIDA traces are shown in figures 8a-8c. Such traces include a large number of multiplexed TCP connections as noted above. The figures show the theoretical number of concurrent open TCP handshakes obtained by NATRA and the actual value of the 0.99 percentile in one-minute time intervals.

The results show that the the approximate capacity planning formulas for the demand of connection records (equation 1) gives an estimate of the size of the list of connection records. In the case of the 2014 trace, this estimate is not very accurate because the flow-arrival process includes peaks that depart from the Poisson distribution. In any case, the estimate will be useful for approximate planning of the system memory capacity, as in equation 2.

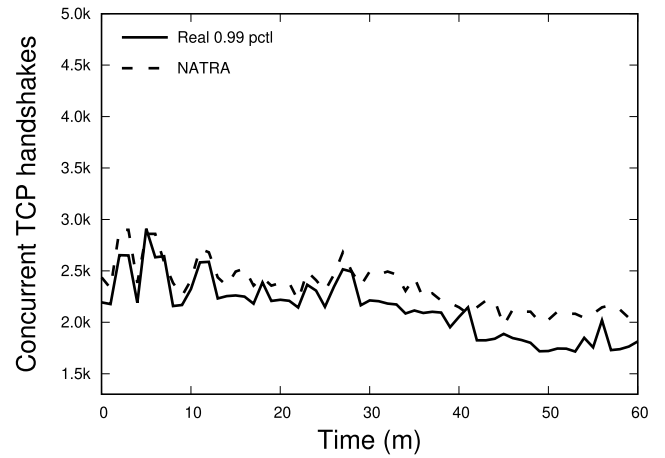
### A. TRAFFIC REDUCTION RATES

Having verified that the number of concurrent TCP handshakes can be approximately upper-bounded using equation 1, we turn our attention to the traffic reduction rate in packets per second. We apply equation 2 with the collision probability  $P = 0.05$  and obtain the memory size  $M$ . Then, we perform a trace-driven simulation and the results are shown in the figures 9-11c, for both memory size in number of slots and the percentage of traffic reduction. The reductions in traffic are remarkable for all the traces considered in our analysis. Furthermore, the memory sizes are compatible with current cache technologies (see section IV-A1), which ensures that multi-Gbps throughput can be attained.

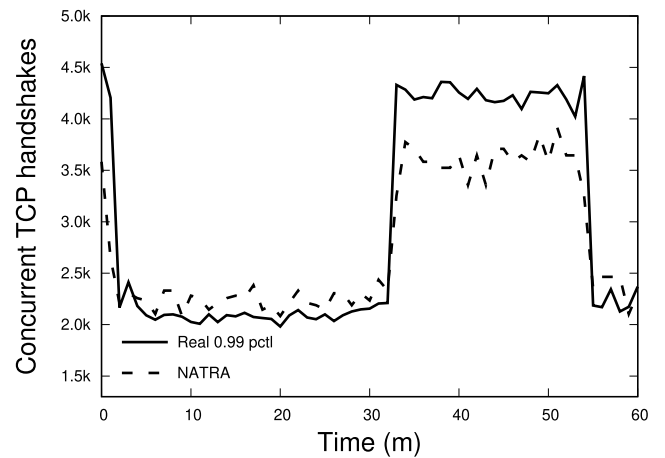
The experimentally measured memory size is larger than the memory size calculated using equation 1 directly, as expected, because the collision probability is non-zero.



(a) CAIDA 2014 Equinix-Chicago traces.



(b) CAIDA 2015 Equinix-Chicago traces.

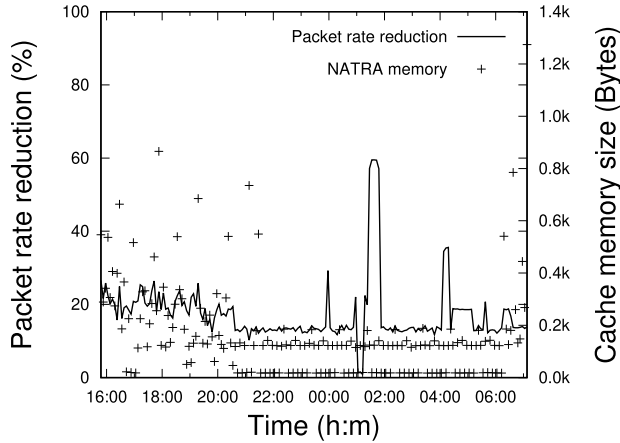


(c) CAIDA 2016 Equinix-Chicago traces.

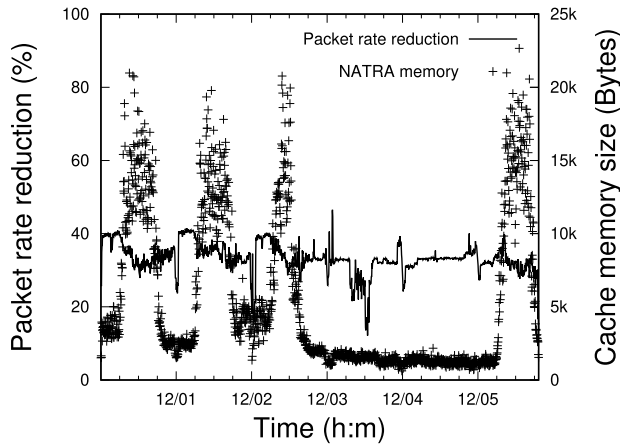
**FIGURE 8.** Concurrent open TCP handshakes.

In any case, we note that *in case of collision, especially if memory is full, the corresponding incoming flow is passed to the upper layers without trimming*. This relationship involves a trade-off between memory occupancy and traffic reduction.





**FIGURE 9.** Percentage of packet rate reduction and memory size for university laboratory traces.



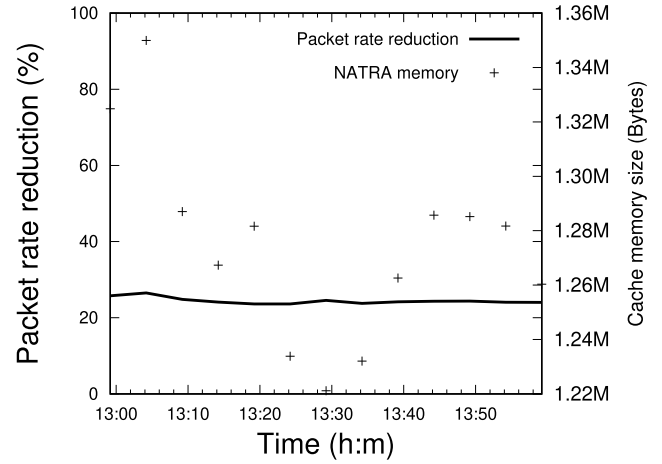
**FIGURE 10.** Reduction in packets per second and memory size for private company traces.

## B. EMPIRICAL EVALUATION WITH A REAL IMPLEMENTATION

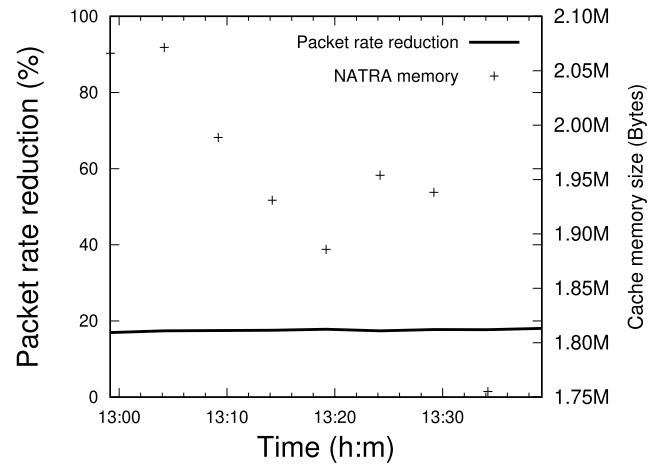
To evaluate the maximum achievable throughput when using NATRA we use our previously reported high-speed driver [18] in user space to achieve a realistic deployment scenario. The driver has one thread in charge of timestamping, which moves all packets from the shared memory to user-space memory. Thus, we implemented NATRA between the timestamping and memory movement tasks.

To make the simulation as realistic as possible, we placed all packets in reverse order into the main memory. This trick bypasses the pre-caching techniques implemented in various operating systems that try to cache as much data stored in memory as possible. As such, programs usually store data sequentially in memory, to make the most of precaching.

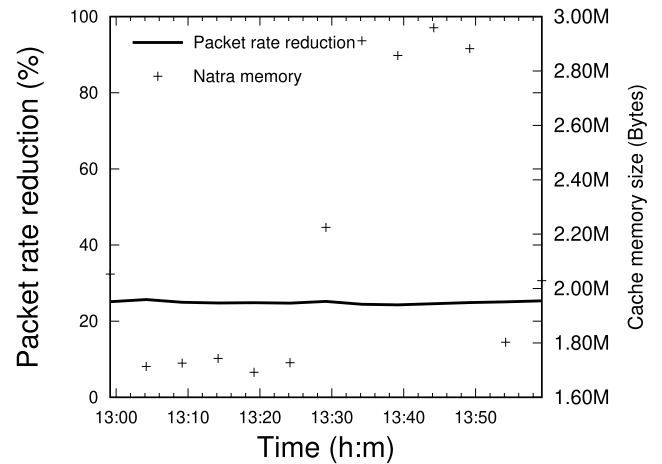
By changing the order of the packets in memory, no pre-caching will be performed because the packets are not stored in well-aligned and sequential memory. In other words, the implementation intentionally forces cache errors when packets are read from the shared memory.



(a) CAIDA 2014 Equinix-Chicago traces.



(b) CAIDA 2015 Equinix-Chicago traces.



(c) CAIDA 2016 Equinix-Chicago traces.

**FIGURE 11.** Reduction percentage and memory size.

The results show that the throughput *per core* reaches 31 Gbps reading from host memory and that the traffic reduction in packets per second is significant up to 25%, which supports the validity of using NATRA for offloading packet sniffers. We stress that *only one core was used in the*

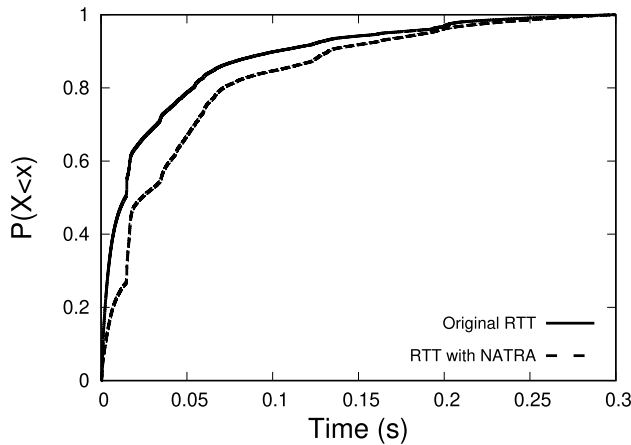


FIGURE 12. RTT comparison for University Internet link traces.

experiment, so the same process could be applied to different RSS queues to speed up the traffic thinning system and reach faster speeds, which is the usual approach for scaling to higher speeds [17], [37].

### C. RTT ESTIMATION ACCURACY AFTER NATRA

As discussed above in section III, NATRA is expected to drop a huge quantity of empty-payload ACK packets. RTT is the KPI most affected by this packet thinning process. Nevertheless, the NATRA process preserves all the packets involved in the handshake process, so the RTT obtained by these packets will not vary.

In any case, we note that RTT may vary during the connection lifetime. Thus, such RTT estimation can be updated by using ACK packets with non-empty payload, which are not removed by NATRA. In what follows, we will assess that the RTT estimation remains accurate, in spite of NATRA removing ACK packets without payload.

As a benchmark, we have measured connection RTT with Tshark [30], which is a well known and widespread tool used for obtaining network performance indicators. Tshark consumes large CPU resources, and it is not well-suited for very big traces. Thus, we used the traces recorded at the university Internet link, which has a smaller size.

Figure 12 shows the Cumulative Density Function (CDF) of the RTTs calculated from the original trace and its NATRA-filtered counterpart. The CDFs are very similar, even though they are not the same because of the lack of ACKs in the latter trace.

Furthermore, we have distinguished between the connections generated to the Internet (Outbound traffic) and the traffic coming from the Internet (Inbound traffic) and have obtained the RTT CDF for each traffic direction. Figure 13 shows the RTTs obtained from inbound connections and figure 14 shows the RTTs obtained from the outbound connections, respectively. As it turns out, inbound connections between original and NATRA-filtered RTT CDFs seems to be closer than the CDFs obtained from the outbound one.

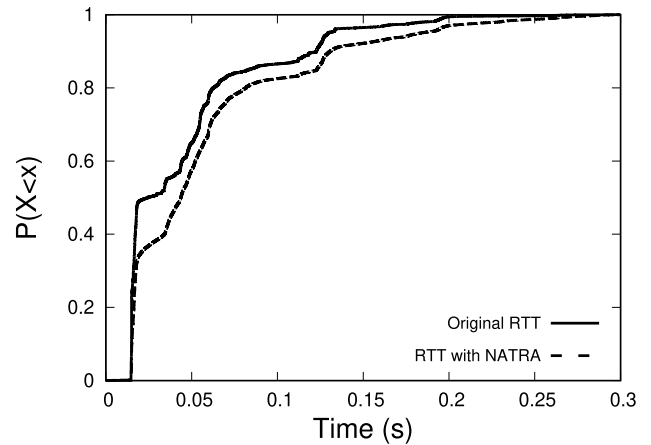


FIGURE 13. RTT comparison (inbound connections) for University Internet link traces.

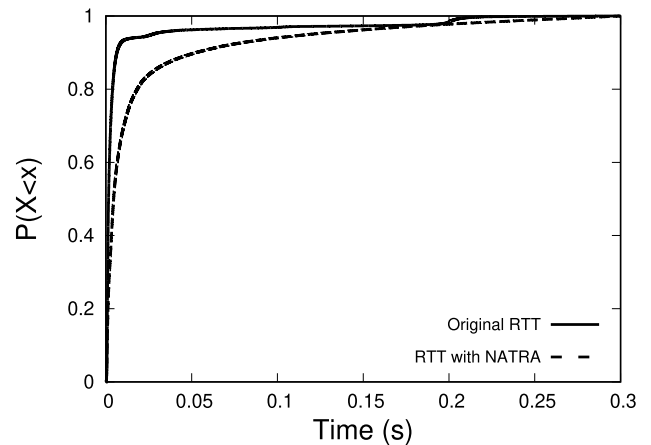


FIGURE 14. RTT comparison (outbound connections) for University Internet link traces.

However, even in the latter case, NATRA provides a good accuracy.

### D. PERFORMANCE GAINS APPLYING NATRA TO WELL-KNOWN TOOLS

To conclude the results and discussion section, we discuss our application of NATRA to the inputs of well-known monitoring tools, and assess the resulting performance gains and accuracy. First, we tested the behaviour of NATRA with FlowMiner, a program used to obtain valuable information from all the different flows included in a PCAP file.

In addition, live monitoring tools have been used to verify how NATRA performs from the CPU load standpoint. The well-known Tstat [3] and Tshark tools were used for these live tests. [1]. We endeavoured to use well-known and free-access tools to ensure the reproducibility of the experiments. Besides, we used Dstat to obtain the system CPU statistics.

#### 1) FlowMiner

Flowminer was used to process the one-hour-long CAIDA traces. This is the only test in this section performed without

**TABLE 3.** Time reduction of using NATRA in Flow Miner with CAIDA traces.

Trace	Without NATRA (seconds)	With NATRA (seconds)	Time reduction %
CAIDA 2014 equinix-chicago	3,599	3,012	16.31
CAIDA 2015 equinix-chicago	4,086	3,674	10.08
CAIDA 2016 equinix-chicago	4,075	3,322	18.47

**TABLE 4.** Differences per flow with and without NATRA on FlowMiner.

Attribute	Values altered (%)
Number of Resets	$2.72 \times 10^{-8}$
Number of data packets	$2.86 \times 10^{-7}$
Number of disorder packets	$1.77 \times 10^{-7}$
Bytes of disorder packets	$1.77 \times 10^{-7}$

live traffic, as Flowminer is an offline processing tool. Note that, since the traces were recorded from Internet backbone links, they include a huge number of concurrent flows, which entails a long processing time. Thus, we assessed NATRA performance in terms of processing and time reduction as a benchmark.

Table 3 lists the total processing times when the original traces and NATRA-filtered ones are given as input to Flowminer. Interestingly, the NATRA-filtered traces can be processed in less than 3600 seconds, which is the exact trace duration. On the contrary, only the 2014 trace could be processed in less than the trace duration without the NATRA filter. The time Flowminer takes to process the traces without NATRA is closer to 4000 seconds. Overall, NATRA leads to a remarkable reduction in processing time, between 10% and 20% depending on the trace.

Packets were reduced by close to 20-25% in the CAIDA traces, but the improvement on performance is not as large. Nevertheless, the traces could be processed in less time than their respective durations, which allows real-time processing.

In addition, as discussed in section V-C, we evaluated to which extent the introduction of NATRA alters the flow record parameter values. Table 6 shows the percentages of flow record parameters that show alteration by using NATRA. The data obtained with and without NATRA is practically the same.

As shown, NATRA accelerates the processing time in Flowminer keeping the validity of the measured metrics.

## 2) TSTAT AND TSHARK

Finally, we ran experiments using the Tstat and Tshark tools with TCPReplay sending live traffic ([2]) in order to assess if CPU performance improves. In this case, the experiment duration remains the same and equal to the traffic replay duration and the CPU load varies by using NATRA.

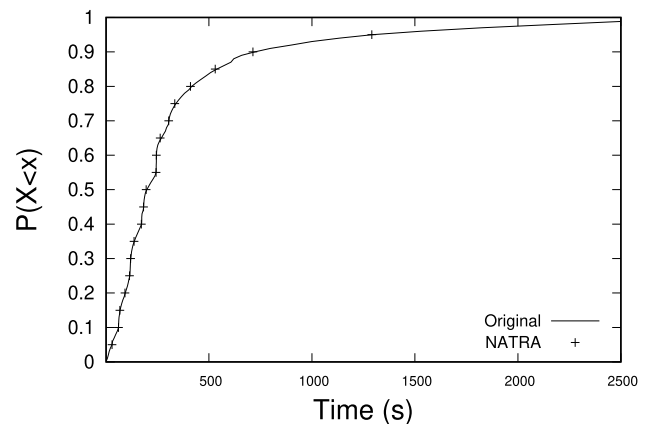
Table 5 shows the CPU load observed, measured in CPU-time employed. Such CPU-time was measured with the Dstat log, once per second, and, then, an average over the experiment lifetime was obtained.

**TABLE 5.** CPU-time reduction obtained with NATRA.

Application	CPU-time reduction %
Tstat	8.80
Tshark	22.92

**TABLE 6.** Differences per flow between NATRA and Original processing on Tshark and Tstat.

Attribute	Values altered (%)
Bytes sent	$3.35 \times 10^{-5}$
Retransmitted segments	$1.42 \times 10^{-6}$
Retransmitted bytes	$4.34 \times 10^{-6}$
Duration(s)	$3.40 \times 10^{-5}$

**FIGURE 15.** Cumulative distribution function of flow duration.

Better results in terms of CPU-time reduction are observed with Tshark. This tool uses a large amount of processing resources and the use of NATRA provides a significant improvement.

Apart from the RTT discussion in section V-C, we have calculated the number of errors in several attributes of the performance of TCP flows using NATRA. We selected different attributes from the ones discussed in section V-D1 for the sake of generality. Similar to the data presented in our discussion of Flowminer, table 6 lists the percentage of values altered for the selected TCP flows' attributes. We note that the percentage of values altered is negligible.

Furthermore, figure 15 compares the flow duration CDF for NATRA and non-NATRA experiments. NATRA causes no noticeable changes in the distribution, but the CPU-time is reduced by nearly 23%. Therefore, NATRA paid off in terms of accuracy and CPU time reduction.

## VI. CONCLUSIONS

This work has presented NATRA, a traffic reduction algorithm that significantly decreases the rate of packets per second that must be analysed by packet sniffers on high-speed networks. In a nutshell, NATRA drops ACK packets that carry no data payload. This offloading gives the sniffer extra CPU time to perform packet processing and analysis. The results from our trace-driven evaluation show traffic reduction rates between 18.70% and 34.54% in packets per second. We assessed the performance of NATRA as a traffic thinner for well-known monitoring tools. The results show remarkable CPU-time reductions, close to 23% using Tshark, 9% using Tstat, and a processing time reduction near 15% using Flowminer, with no information loss.

Large reductions in traffic rates can be achieved, which reduce the processing load involved in subsequent analysis and visualization stages beyond packet capture. This is cornerstone to scale network traffic analysis at tens of Gbps speeds.

## REFERENCES

- [1] *Dstat Source Code Download Web Access*. Accessed: Jul. 26, 2020. [Online]. Available: <https://github.com/dagwieers/dstat>
- [2] *Tcpreplay Source Code Download Web Access*. Accessed: Jul. 26, 2020. [Online]. Available: <https://tcpreplay.appneta.com/>
- [3] *Tstat Source Code Download Web Access*. Accessed: Jul. 26, 2020. [Online]. Available: <http://tstat.polito.it/download/tstat-3.0.1.tar.gz>
- [4] K. Accardi, T. Bock, F. Hady, and J. Krueger, "Network Processor Acceleration for a Linux Netfilter Firewall," in *Proc. ACM Symp. Archit. Netw. Commun. Syst.*, New York, NY, USA, 2005, pp. 115–123.
- [5] M. Allman, V. Paxson, and W. Richard Stevens, *TCP congestion control*, document RFC 2581, 1999.
- [6] T. Alonso, M. Ruiz, G. Sutter, S. López-Buedo, and J. L. de Vergara Méndez, "Towards 100GbE FPGA-based flow monitoring," in *Proc. 10th Southern Program. Logic Conf.*, Buenos Aires, Argentina, Apr. 2019, doi: 10.1109/SPL.2019.8714532.
- [7] J. Z. Apisdorf, S. B. Sandbote, and M. D. Poole, "System and method for instruction-level parallelism in a programmable multiple network processor environment," U.S. Patent 11 862 815, Apr. 2, 2013.
- [8] F. Arts, P. Barri, I. Clemminck, A. Niemegeers, B. Pauwels, G. Taideman, and M. Vrana, "Network processor requirements and benchmarking," *Comput. Netw.*, vol. 41, no. 5, pp. 549–562, 2003.
- [9] N. Bonelli, S. Giordano, and G. Procissi, "Network traffic processing with PFQ," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 6, pp. 1819–1833, Jun. 2016.
- [10] (2009). *Analyzing UDP usage in Internet traffic*. [Online]. Available: <https://www.caida.org/research/traffic-analysis/tcpudpratio/>
- [11] (2014). *The CAIDA UCSD Anonymized Internet Traces 2014, 2015 and 2016*. [Online]. Available: [http://www.caida.org/data/passive/passive\\_dataset.xml](http://www.caida.org/data/passive/passive_dataset.xml)
- [12] J. Cathey and T. S. Michels, "Programmable packet processor with flow resolution logic," U.S. Patent 8 724 632, May 13, 2014.
- [13] T. T. Cheng, "The normal approximation to the Poisson distribution and a proof of a conjecture of Ramanujan," *Bull. Amer. Math. Soc.*, vol. 55, no. 4, pp. 396–401, 1949.
- [14] P. Emmerich, M. Pudelko, S. Gallenmüller, and G. Carle, "FlowScope: Efficient packet capture and storage in 100 Gbit/s networks," in *Proc. IFIP Netw. Conf. (IFIP Networking) Workshops*, Jun. 2017, pp. 1–9.
- [15] M. Fekrou, T. Houdoin, B. Le Guyader, J. De Biasio, A. Gravey, and J. Alfonso Torrijos Gijón, "Internet traffic analysis: A case study from two major European operators," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jun. 2014, pp. 1–7.
- [16] M. Fomenkov, K. Keys, D. Moore, and K. Claffy, "Longitudinal study of Internet traffic in 1998–2003," in *Proc. Winter Int. Symposium Inf. Commun. Technol.*, 2003, pp. 1–6.
- [17] J. L. García-Dorado, F. Mata, J. Ramos, M. P. S. del Río, V. Moreno, and J. Aracil, *High-Performance Network Traffic Processing Systems Using Commodity Hardware*. Berlin, Germany: Springer, 2013, pp. 3–27.
- [18] (2014). *HPCAP Driver GIT Repository*. [Online]. Available: <https://github.com/hpcn-uam/HPCAP>
- [19] K. Hu, H. Kumarapillai Chandrikakutty, Z. Goodman, R. Tessier, and T. Wolf, "Dynamic hardware monitors for network processor protection," *IEEE Trans. Comput.*, vol. 65, no. 3, pp. 860–872, Mar. 2016.
- [20] J. Iyengar and M. Thomson, "QUIC: A UDP-based multiplexed and secure transport," Internet Eng. Task Force, Fremont, CA, USA Tech. Rep. draft-ietf-quic-transport-19, Mar. 2019. [Online]. Available: <https://tools.ietf.org/html/draft-ietf-quic-transport-19>
- [21] M. Kihl, P. Ödling, C. Lagerstedt, and A. Aurelius, "Traffic analysis and characterization of Internet user behavior," in *Proc. Int. Congr. Ultra Modern Telecommun. Control Syst.*, Oct. 2010, pp. 224–231.
- [22] R. Leira, G. Julián-Moreno, I. González, and F. J. Gómez-Arribas, "Performance assessment of 40 gbit/s off-the-shelf network cards for virtual network probes in 5g networks," *Comput. Netw.*, vol. 152, pp. 133–143, Oct. 2019.
- [23] M. Liu, T. Cui, H. Schuh, A. Krishnamurthy, S. Peter, and K. Gupta, "Offloading distributed applications onto smartnics using ipipe," in *Proc. ACM Special Interest Group Data Commun.*, Aug. 2019, pp. 318–333.
- [24] A. Mahmood and T. Sauter, "Improving the accuracy of software-based clock synchronization and encountering interrupt coalescence," in *Proc. IEEE Int. Symp. Precis. Clock Synchronization for Meas., Control, Commun. (ISPCS)*, Oct. 2015, pp. 82–87.
- [25] M. Mellia, M. Meo, L. Muscariello, and D. Rossi, "Passive analysis of TCP anomalies," *Comput. Netw.*, vol. 52, no. 14, pp. 2663–2676, 2008.
- [26] E. Miravalls-Sierra, D. Muelas, J. Ramos, E. J. L. de Vergara, D. Morató, and J. Aracil, "Online Detection of Pathological TCP Flows with Retransmissions in High-speed Networks," *Comput. Commun.*, vol. 127, pp. 95–104, Sep. 2018.
- [27] V. Moreno, J. Ramos, P. M. Santiago del Río, J. Luis García-Dorado, F. J. Gomez-Arribas, and J. Aracil, "Commodity packet capture engines: Tutorial, cookbook and applicability," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 3, pp. 1364–1390, thi. 2015.
- [28] V. Moreno, P. M. S. D. Río, J. Ramos, D. Muelas, J. L. García-Dorado, F. J. Gomez-Arribas, and J. Aracil, "Multi-granular, multi-purpose and multi-Gb/s monitoring on off-the-shelf systems," *Int. J. Netw. Manage.*, vol. 24, no. 4, pp. 221–234, 2014.
- [29] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "Trumpet: Timely and precise triggers in data centers," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 129–143.
- [30] A. Orebaugh, G. Ramirez, and J. Beale, *Wireshark & Ethereal Network Protocol Analyzer Toolkit*. Amsterdam, The Netherlands: Elsevier, 2006.
- [31] S. Phillipa and M. Anirban, "Observations on round-trip times of TCP connections," in *Proc. Int. Symp. Perform. Eval. Comput. Telecommun. Syst. (SPECTS)*, 2006, p. 347.
- [32] P. M. Photilimthana, M. Liu, A. Kaufmann, S. Peter, R. Bodik, and T. Anderson, "Floem: A programming system for nic-accelerated network applications," in *Proc. 12th USENIX Conf. Operating Syst. Design Implementation*, 2018, pp. 663–679.
- [33] 2017. *Sandra Benchmark Suite*, SiSoftware. [Online]. Available: <http://www.sisoftware.eu>
- [34] L. Sihyung, L. Kyriaki, and S. Kim Hyong, "Network monitoring: Present and future," *Comput. Netw.*, vol. 65, pp. 84–98, Oct. 2014.
- [35] 2015. *SPDK LIB GIT Repository*. [Online]. Available: <https://github.com/spdk/spdk>
- [36] (2015). *Storage Performance Development Kit WEB Page*. [Online]. Available: <http://www.spdk.io/>
- [37] M. Trevisan, A. Finamore, M. Mellia, M. Munafo, and D. Rossi, "Traffic analysis with off-the-shelf hardware: Challenges and lessons learned," *IEEE Commun. Mag.*, vol. 55, no. 3, pp. 163–169, Mar. 2017.
- [38] X. Wang, M. Veeraraghavan, M. Brandt-Pearce, T. Miyazaki, N. Yamanaka, S. Okamoto, and I. Popescu, "A dynamic network design for high-speed enterprise access links," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2015, pp. 1–7.
- [39] M. Yu, A. G. Greenberg, D. A. Maltz, J. Rexford, L. Yuan, S. Kandula, and C. Kim, "Profiling network performance for multi-tier data center applications," in *Proc. 8th USENIX Conf. Netw. Syst. Des. Implement. NSDI*, vol. 11, 2011, p. 5.
- [40] J. F. Zazo, M. Forconesi, S. Lopez-Buedo, G. Sutter, and J. Aracil, "TNT10G: A high-accuracy 10 GbE traffic player and recorder for multi-terabyte traces," in *Proc. Int. Conf. ReConfigurable Comput.*, Dec. 2014, pp. 1–6.

- [41] T. Zhang, L. Linguaglossa, M. Gallo, P. Giaccone, and D. Rossi, "FloWatcher-DPDK: Lightweight line-rate flow-level monitoring in software," *IEEE Trans. Netw. Service Manage.*, vol. 16, no. 3, pp. 1143–1156, Sep. 2019.
- [42] B. Wang, Z. Zhang, and J. Lan, "Identifying elephant flows in Internet backbone traffic with Bloom filters and LRU," *Comput. Commun.*, vol. 61, no. 1, pp. 70–78, 2015.



**SANTIAGO GARCIA-JIMENEZ** (Member, IEEE) received the M.Sc. and Ph.D. degrees in computer science from the Public University of Navarra, Pamplona, Spain, in 2007 and 2013, respectively. He had a fellowship with the Telecommunications Engineering Department, from 2003 to 2008. He has participated in several European projects. His research interests include network monitoring based on active and passive measurements, network measurement platforms, discovery of Internet topology, IP addresses alias resolution, and web troubleshooting.



**EDUARDO MAGAÑA** (Member, IEEE) received the M.Sc. and Ph.D. degrees in telecommunications engineering from the Public University of Navarra, Pamplona, Spain, in 1998 and 2001, respectively. He is currently an Associate Professor at the Public University of Navarra. In 2002, he was a Visiting Postdoctoral Research Fellow at the Department of Electrical Engineering and Computer Science, University of California at Berkeley. His main research interests include network monitoring, traffic analysis, and performance evaluation of communication networks.



**JAVIER ARACIL** (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees (Hons.) in telecommunications engineering from the Technical University of Madrid, in 1993 and 1995, respectively. In 1995, he was appointed as a Postdoctoral Researcher at the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley. In 1998, he was a Research Scholar at the Center for Advanced Telecommunications, Systems and Services, The University of Texas at Dallas. He has been an Associate Professor of the University of Cantabria and the Public University of Navarra. He is currently a Full Professor at the Universidad Autónoma de Madrid, Madrid, Spain. His research interests include optical networks and performance evaluation of communication networks. He has authored more than 100 papers in international conferences and journals. In 1995, he received the Fulbright Scholarship.

...